J Ambient Intell Human Comput (2017) 8:47–55
DOI 10.1007/s12652-016-0344-9

ORIGINAL RESEARCH

Mining frequent items and itemsets from distributed data streams for emergency detection and management

Albino Altomare[1] · Eugenio Cesario[1] · Domenico Talia[2]

Received: 26 June 2015 / Accepted: 12 January 2016 / Published online: 29 January 2016

Abstract Sensor networks are an important technology for large-scale monitoring, that allow the collection of environmental measurement streaming data in remote areas. Such data constitute a valuable source of information to be exploited for better understanding natural phenomena. Moreover, in some cases streams of data must be analyzed in real time to provide information about trends, outlier values or regularities that must be signaled as soon as possible, to prevent emergencies or disasters (e.g., landslides, fires). For such a reason, real-time analysis of distributed data streams is a challenging task since it requires scalable solutions to handle streams of data that are generated very rapidly by multiple sources. This paper presents the design and the implementation of an architecture for the analysis of data streams in distributed environments. Experimental evaluation shows the efficiency and effectiveness of the approach.

1 Introduction

Big amounts of complex data are available from environmental systems and constitute a valuable source of information to be exploited for better understanding natural phenomena. Such data can be collected by a large variety of sensors that the commercial remote sensing industry provides. For example, many companies are very active in the design and development of sensors for remote sensing of natural resources (vegetation, water, impervious surfaces, and soil), as well as water and energy fluxes, clouds, atmospheric pollutants, surface temperature, etc., with the goal of understanding physical, ecological, hydrological and environmental characteristics of surfaces and substances. This is leading to an exponential growth of environmental data volumes.

Sensor networks, for example, are an important technology for large-scale monitoring, that allow the collection of environmental measurement streaming data (soil moisture, leaf wetness, solar radiation, barometric pressure, humidity, temperature, etc.) in remote areas. By such technological infrastructures, a huge quantity of data, which is still growing very rapidly both in the volume and complexity, is generated. For such a reason, the analysis of environmental data requires advanced and sophisticated modeling techniques to extract the proper relevant knowledge, and increasingly perform real-time data processing or aggregation to provide useful summary information to the domain experts. In particular, frequent pattern discovery is one of the most interesting issues in environmental analysis, because it is aimed at detecting associations, correlations and causality relations between value measurements, and it can reveal insights into natural dynamics. Moreover, in some cases streams of data must be analyzed in real time to provide information about trends, outlier values or regularities that must be signaled as soon as possible, to prevent emergencies or disasters (e.g., landslides, fires). So, real-time analysis of distributed data streams is a challenging task since it requires scalable solutions to handle streams of data that are generated very rapidly by multiple sources.

✉ Eugenio Cesario
cesario@icar.cnr.it

Albino Altomare
altomare@icar.cnr.it

Domenico Talia
talia@dimes.unical.it

[1] ICAR-CNR, Via P. Bucci 7-11 C, 87036 Rende, CS, Italy

[2] DIMES-UNICAL, Via P. Bucci 41C, 87036 Rende, CS, Italy

This work deals with the design and the implementation of an architecture for the analysis of data streams in distributed environments. In particular, data stream analysis has been carried out for the computation of items and itemsets that exceed a frequency threshold. The mining approach is hybrid, that is, frequent items are calculated with a single pass, using a sketch algorithm, while frequent itemsets are calculated by a further multi-pass analysis. The architecture combines parallel and distributed processing to keep the pace with the rate of distributed data streams. In order to keep computation close to data, miners are distributed among the domains where data streams are generated. The algorithm and the prototype of the architecture have been tested on a real environmental dataset storing measurements collected by 30 sensors distributed on a mountain area of Italy, aimed at fire emergency detection and management. Moreover, we present also a scalability analysis obtained by performing several experiments on a network composed of three domains.

The paper is structured as follows. Section 2 reports the state of the art on distributed frequent pattern mining. Section 3 summarizes the main issues to be taken into account in mining data streams and illustrates the most common algorithms for the computation of frequent items and itemsets. Section 4 describes the proposed parallel/distributed architecture for mining data streams and discusses the adopted hybrid approach. Section 5 reports the results of the algorithm on the a real-life dataset, as well as a scalability analysis of the architecture. Section 6 concludes the paper.

2 Related work

The approaches proposed in literature to discover frequent items and itemsets from data streams can be divided in three main classes. The first class includes the *counter-based* algorithms, that have their foundation on some techniques proposed to solve the *majority* problem (Fisher et al. 1982). *LossyCounting* is one of the most popular algorithm of this type (Manku et al. 2002). The second class refers to the *sketch-based* algorithms, which compute a linear projection of the input (i.e., named *sketch*), and provide an approximated estimation of item frequencies using limited computing and memory resources. Popular algorithms of this kind are *CountSketch* (Charikar et al. 2002) and *CountMin* (Cormode et al. 2005), and the latter is adopted in this paper. A third class includes the *Quantile* algorithms, where the problem of finding the ϕ-quantiles of a sequence of items drawn from a totally ordered domain is to find an item i such that it is the smallest item which dominates $\phi \cdot N$ items from the input. A quantile summary consists of small number of data points from the original data sequence and use this information to calculate any quantile queries. *GK* (Greenwald et al. 2001) and *QDigest* (Shrivastava et al. 2004) are the most famous algorithms belonging to the *quantile* category. Even if modern single-pass algorithms are extremely sophisticated and powerful, multi-pass algorithms are still necessary either when the stream rate is too rapid, or when the problem is inherently related to the execution of multiple passes, which is the case, for example, of the frequent itemsets problem. Single-pass algorithms can be forced to check the frequency of 2- or 3-itemsets, but this approach cannot be generalized easily, as the number of candidate k-itemsets is combinatorial, and it can become very large when increasing the value of k (Jin et al. 2005). Therefore, a very promising avenue could be to devise hybrid approaches, which try to combine the best of single- and multiple-pass algorithms (Wright 2010). The analysis of streams is even more challenging when data is produced by different sources spread in a distributed environment, like a large-scale monitoring system of environmental data. A thorough discussion of the approaches currently used to mine multiple data streams can be found in (Parthasarathy et al. 2007). The paper distinguishes between the *centralized* model, under which streams are directed to a central location before they are mined, and the *distributed* model, in which distributed computing nodes perform part of the computation close to the data, and send to a central site only the models, not the data. Of course, the distributed approach has notable advantages in terms of degree of parallelism and scalability. An interesting approach for the continuous tracking of complex queries over collections of distributed streams is presented in Cormode et al. (2008). To reduce the communication overhead, the adopted strategy combines two technical solutions: (1) remote sites only communicate to the coordinator concise summary information on local streams (in the form of sketches); (2) even such communications are avoided when the behavior of local streams remains reasonably stable, or predictable: updates of sketches are only transmitted when a certain amount of change is observed locally.

3 Mining frequent items and frequent itemsets in distributed data streams

A data stream (Gaber et al. 2005) is defined as the continuous arrival of data items to a computational device, which must perform most processing analysis online, because the amount of data is so large that its complete archiving is too costly. As mentioned, the discovery of *frequent items* (Cormode et al. 2008) is a very important task, consisting of identifying the items whose frequency in a stream exceeds a specified fraction σ of the overall stream size. The problem is formalized as follows (Cormode et al. 2009):

Problem statement. *Given a stream S of n items* $e_1, \ldots, e_n$, *where the frequency of an item i is* $f_i = |\{j|e_j = i\}|$, *and a frequency threshold* σ, *the* ϵ*- approximate-frequent-items problem consists of finding the set F of items such that*: $F = \{i|f_i \geq (\sigma - \epsilon)n\}$

Two basic categories of algorithms can be used to solve this problem: the *Counter-Based* and the *Sketch-Based* algorithms. Algorithms in the first group maintain counters for a subset of elements, and counters are updated every time one of these elements is observed in the stream. If the observed element has no associated counter, the algorithm must choose whether to ignore the element or replace an existing counter with a counter for the new item. At the end of the first pass, frequent items will surely be among those associated with counters, but the inverse is not true, which requires at least a second pass to verify which counters actually correspond to frequent items. Conversely, *Sketch-Based* algorithms (Cormode et al. 2008) do not monitor a subset of elements but provide, with a given accuracy, an estimation of the frequency for all stream elements using a matrix of counters C with d rows and w columns. A set of d hash functions $h_1, \ldots, h_d$ are chosen among a family of *pairwise-independent* functions, and are associated to the different matrix rows. Each item i observed in the stream is mapped, for each row r, to the matrix element $C[r, h_r(i)]$. The sketch-based *CountMin* algorithm (Cormode et al. 2005), at the arrival of a new item i, update the counter as follows:

$$for\ r \in [1, d] \rightarrow C[r, h_r(i)] += 1$$

The number of counters in a row, w, is lower than the number of elements, so there are conflicts, because several distinct elements will be mapped by a hash function to the same counter. However, different elements are in conflict for different rows, which enables the adoption of statistical techniques to estimate the actual frequencies of elements. In *CountMin*, collisions always cause extra increments of counters, therefore the best estimation for the frequency f_i of element i is the minimum value of the counters associated to i: $f_i = min_r\{C[r, h_r(i)]\}$. It is worth recalling that the overall sketch of multiple streams can be computed by adding the sketches of single streams. This is the main reason why we decided to adopt *CountMin*: in a distributed architecture, it would be prohibitive to transmit source data to a central processing node, while the mere transmission of sketch summaries allows communication overhead to be drastically reduced.

4 A hybrid multi-domain architecture

This section presents the stream mining architecture that aims at solving the problem of computing frequent items and frequent itemsets from distributed data streams. The architecture includes the following components (Fig. 1):

- *Data Streams* (*DS*) data source nodes, located in different domains.
- *Miners* (*M*) placed close to the respective Data Streams, miners perform two basic mining tasks: the computation of sketches for the discovery of frequent items, and the computation of the support count of candidate frequent itemsets. Each Miner computes the sketch only for the data it receives, and then forwards the results to the local Stream Manager.
- *Stream Managers* (*SM*) in each domain, the Stream Manager collects the sketches computed by local miners, and derives the sketch for the local DS. Moreover, each SM cooperates with the Stream Manager Coordinator to compute global statistics, valid for the union of all the Data Streams.
- *Stream Managers Coordinator* (*SMC*) this node collects mining models from different domains and computes overall statistics regarding frequent items and frequent itemsets. The SMC can coincide with one of the Stream Managers, and can be chosen with an election algorithm.
- *Data Cachers* (*DC*) are essential to enable the hybrid strategy. Each Data Cacher stores the statistics about frequent items discovered in the local domain. These results are then re-used by Miners to discover frequent itemsets composed of increasing numbers of items.

The algorithm for the computation of frequent items, outlined in Fig. 2 (left side), is performed continuously, for every new block of data that is generated by the data streams. A *block* is defined here as the set of transactions that are generated in a time interval P. If the generation rate is too fast to be sustained by a single Miner, a filter is used to partition the block into as many mini-blocks as the number of available miners (step 1 in the figure). Each Miner computes the sketch related to the received mini-block (step 2) and transmits it to the SM (step 3), which overlaps the sketches, thanks to the linearity property of sketch algorithms, and extracts the frequent items for the local domain. Then, every SM sends the local sketch to the SMC (step 4a) and the Miners send the most recent blocks of transactions to the local Data Cacher (step 4b). At step 5, the SMC aggregates the sketches received by SMs and identifies the items that are frequent for the union of data streams. Frequent items are computed for a *window* containing the most recent W blocks. This can be done easily thanks to the linearity of the sketch algorithm: at the arrival of a new block, the sketch of this block is added to the current sketch of the window, while the sketch of the least recent block is subtracted.

The schema of the algorithm for mining frequent itemsets is illustrated in Fig. 2 (right side), assuming that steps 1–5 (frequent items computation) have already

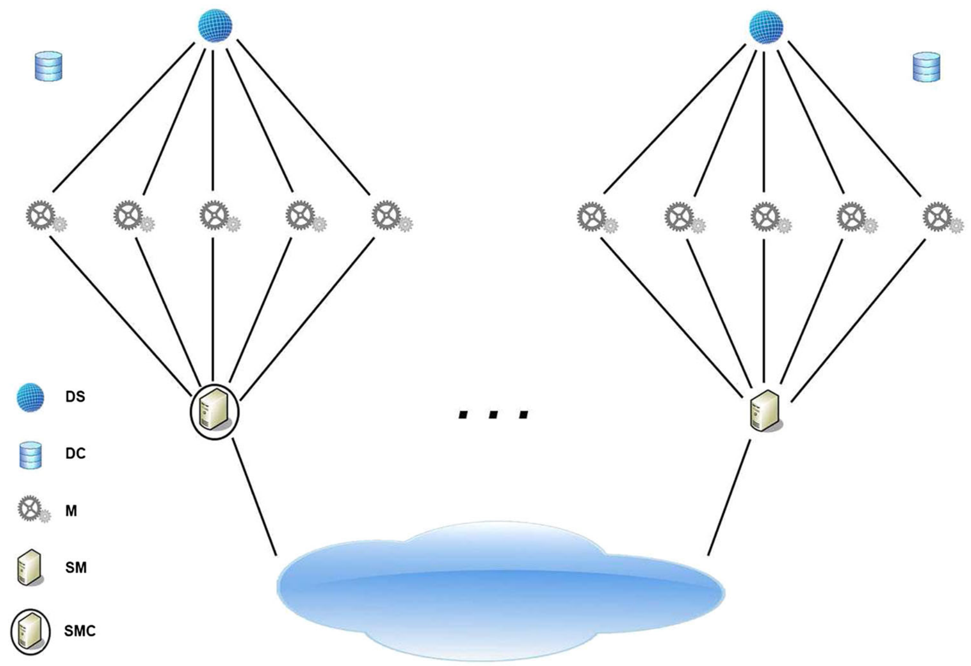


Fig. 1 Distributed architecture for data stream mining

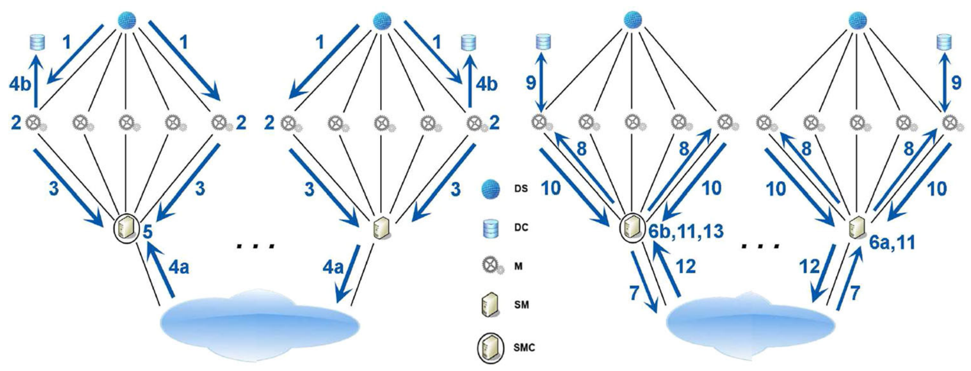


Fig. 2 Schema of the algorithm for mining frequent items (*left*) and itemsets (*right*)

been performed. At step 6, every SM builds the candidate k-itemsets for the local domain (6a), and the SMC also builds the global candidate k-itemsets (6b). The SMC sends the global candidates to the SMs for the computation of their support at the different domains (step 7). The SMs send both local and global candidates to the Miners (step 8), which turn to the Data Cacher to retrieve the transactions included in the current window (step 9),[1] compute the support count for all the candidates, and transmit the results to the local SM (step 10). The SM aggregates the support counts received by Miners and selects the k-itemsets that are frequent in the local domain (step 11). Analogously, the SMs send the SMC the support counts of the global candidates (step 12), and the SMC computes the itemsets that are frequent over the whole system (step 13). The algorithm restarts from step 6 to find frequent itemsets with increasing numbers of items. The cycle stops either when the maximum allowed size of itemsets is reached or when no frequent itemset was found in the last iteration.

5 Prototype and experimental evaluation

The architecture described in the previous section was implemented using the *Mining@Home* system, adopted to perform several classes of data mining computations (Cesario et al. 2014, 2009). Section 5.1 shows the effectiveness of the algorithm, by describing the results obtained on a real-life dataset. Section 5.2 presents the efficiency

[1] Miners may have the ability to store some transactions in their own memory. In this case, they only ask the Data Cacher those transactions that could not be stored locally.

analysis of the approach, obtained on reference dataset. The parameters used to assess the prototype are listed below:

- P the time interval to receive a block of data. This interval determines the average number of transactions generated within a block, denoted as N_t, and the average size of a block in bytes, B;
- N_{MD} the number of available miners per domain. In our experiments, this number is the same for the three domains;
- N_M the total number of available miners in the Grid, equal to $3 \cdot N_{MD}$;
- F_{CPU} the fraction of CPU time reserved on miners for the experiments, set to 30 %. This setting was used to make the results independent from the execution of other processes on the same nodes.[2]
- S the support threshold used to determine frequent items and itemsets;
- W the size of the sliding window, i.e., the number of consecutive blocks of data on which computation is performed;
- C_M the capacity of the miner buffer. Unless otherwise stated, it is equal to the size of a data block B.
- ϵ and δ, the accuracy parameters of the sketch algorithm (both set to 0.01).

5.1 A real-world case study: analysis of the ‘AltaIrpinia’ dataset

In this section we report the results obtained by using the proposed architecture on the *AltaIrpinia* data, a real-life dataset populated by several environmental measures of a mountainous area of Italy. This dataset is a collection of data streams collected by 30 sensors distributed on the territory of Avellino (Campania, South Italy), whose morphology is presented in Fig. 3. The list of the monitoring stations (which sensors are positioned on) as well as their localizations is reported in Table 1. The dataset AltaIrpinia contains several measurements collected from June 2013 to November 2013. Sensors have detected, with a sampling period of twenty minute, four environmental values: *leaf wetness* (%), *soil moisture* (%), *atmospheric temperature* (°C) and *soil temperature* (°C). The total number of values was fifty thousands circa, where data size amounted to almost 21 MB. Such data measurements, completed by the *LocationID*, *StationID* and *Timestamp* features, have been stored and analyzed in streaming, to discover in real-time some environmental frequent patterns as well as to quickly provide information about trends, outlier values, regularities hidden in the data. Such knowledge is very useful to detect fire threats that must be signaled as soon as possible, for efficient emergency detection and management.

As a preliminary step for the analysis, all the values have been properly discretized by a pre-processing step in several bins, representing different ranges of values. To do that, we adopted a unsupervised discretization approach. The analysis has been performed on 30 domains, one data source (a sensor) and one miner running on each one. We fixed $S = 2\,\%$ and $B = 400$, while all other parameters assumed default values. Such experiments were aimed at computing both items and itemsets that are frequent for a single domain and those that are frequent for the union of distributed streams. Table 2 reports some frequent itemsets of length 2, 3 and 4, as well as their support values, returned by the algorithm. Interestingly, a very regular association between measurements can be observed in the 4-frequent patterns, which highlight that usually leaf wetness and soil moisture assume low values (around 0 and 10 %, respectively), while the atmospheric temperature and soil temperature are around 11.5 and 6 °C, respectively. Such trends are confirmed, with an higher support, for itemsets of length 3 and 2. So, outlier values must be signaled as soon as possible, to prevent emergencies or disasters (e.g., fire, landslides).

The whole AltaIrpinia dataset has been analyzed in about 5 s. Our experiments showed that the architecture is suitable to analyze such kind of data, but the size of that dataset is limited to perform a scalability analysis on it. However, it is worth noting that Sect. 5.2 reports a detailed analysis about scalability and efficiency of the system, carried on a larger dataset (i.e., 1 M instances) that is more appropriate to evaluate computational issues.

5.2 Scalability analysis of the architecture

To assess an efficiency analysis of the prototype, we used the transactional dataset published by the *“kosarak”* of the *FIMI Repository*, composed of 1 millions of instances and appropriate for a scalability analysis. More in detail, such a dataset contains a list of click-streams generated by users of an online portal and its analysis is useful to identify the most popular sections of the portal, the preferences and requirements of users, etc. During the experiments, data (stored on the Data Source nodes) are sent to local Miners with a specified transfer rate, to reproduce the original data stream. The transmission rate is adjusted by setting the parameter N_t, the number of transactions generated during the time interval P.

The main performance index assessed during the experiments is the *execution time*, defined as the time interval between the transmission of a new block of stream

[2] the fraction of CPU is tuned using the program *“cpulimit”*.

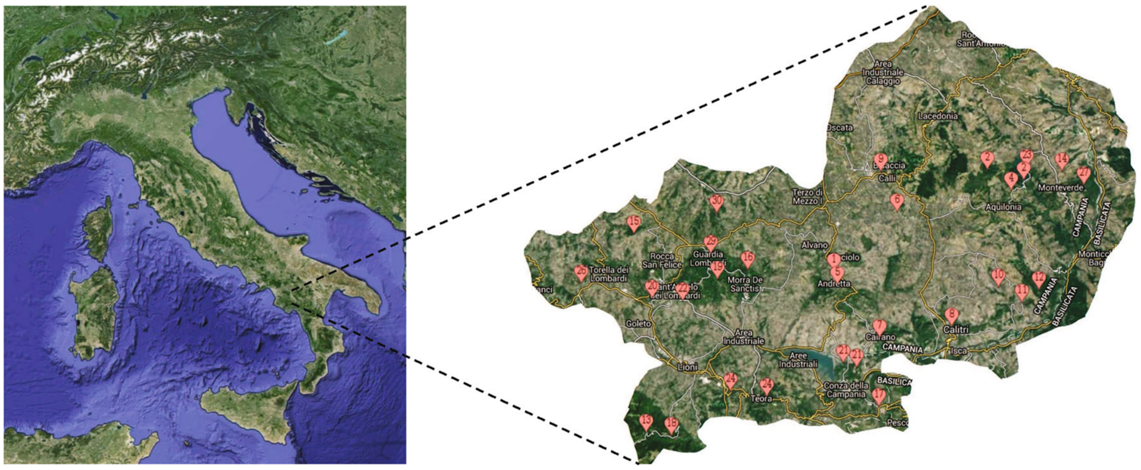


Fig. 3 Areas of the territory monitored for the fire detection

Table 1 Station ID, longitude, latitude and altitude of the sensor stations

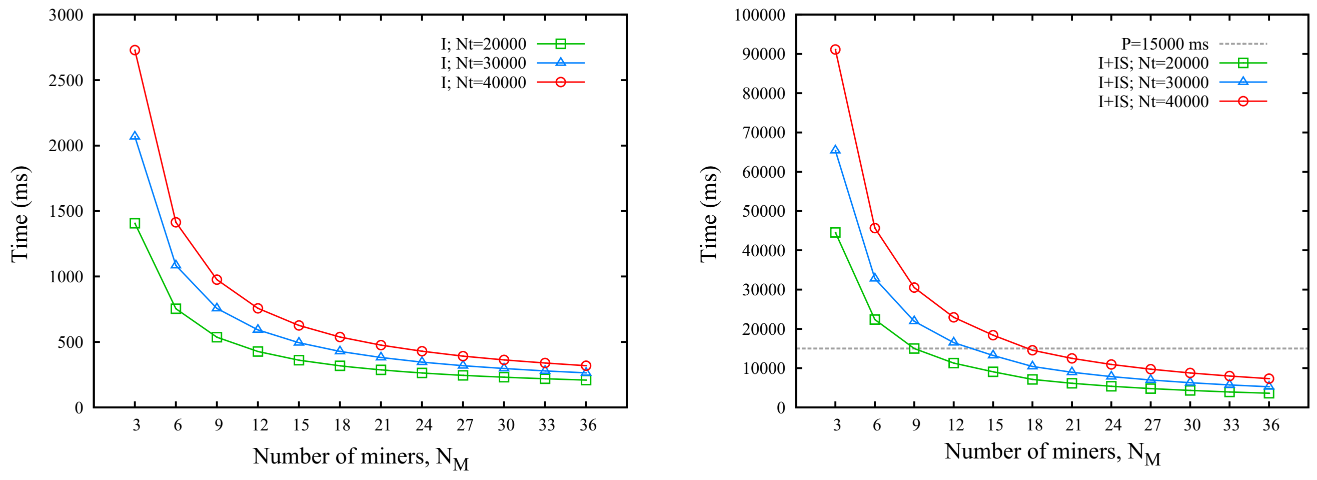
ID	Long	Lat	Alt	ID	Long	Lat	Alt
1	15°20′41.73″E	40°57′3.40″N	820	16	15°12′56.58″E	40°58′51.57″N	875
2	15°19′43″94E	40°55′53.54″N	875	17	15°9′20.88″E	40°49′40.98″N	1053
3	15°30′2.65″E	41°0′14.76″N	473	18	15°10′36.12″E	40°49′30.18″N	957
4	15°29′20.14″E	40°59′27.13″N	473	19	15°30′21.18″E	41°0′33.00″N	461
5	15°28′41.04″E	41°0′43.38″N	545	20	15°33′25.83″E	41°0′5.60″N	423
6	15°21′54.91″E	41°0′37.34″N	825	21	15°31′46.80″E	40°59′43.56″N	757
7	15°24′24.40″E	40°59′47.38″N	830	22	15°14′38.52″E	40°56′37.56″N	778
8	15°22′6.31″E	40°53′32.56″N	625	23	15°8′25.00″E	40°58′0.60″N	735
9	15°26′1.29″E	40°54′12.29″N	434	24	15°21′44.27″E	40°50′51.00″N	590
10	15°30′54.40″E	40°55′37.49″N	564	25	15°12′57.78″E	40°56′6.18″N	875
11	15°29′54.18″E	40°56′27.72″N	542	26	15°9′23.70″E	40°55′17.22″N	673
12	15°29′55.00″E	40°54′40.66″N	496	27	15°8′12.56″E	40°54′53.71″N	690
13	15°20′44.40″E	40°52′20.76″N	444	28	15°13′42.42″E	40°51′17.50″N	557
14	15°19′42.60″E	40°52′30.78″N	474	29	15°15′46.93″E	40°51′8.96″N	742
15	15°12′37.38″E	40°57′6.12″N	1016	30	15°4′51.81″E	40°55′15.87″N	675

data and the time at which the analysis of this block has been completed by the Stream Manager Coordinator (SMC). If this value is not longer than the time interval P, it means that the system is able to keep the pace with data production. The experiments were executed assuming a time period P equal to 15 s. As generation rates, we considered values of N_t equal to about 40,000, 30,000 and 20,000 transactions per block. The generation rates were equally partitioned between the three domains. These are very high generation rates, and allowed the prototype to be tested in challenging conditions.

Figure 4 reports the execution time experienced for the computation of frequent items exclusively (I), and for the computation of both frequent items and itemsets (I+IS), vs. the total number of miners N_M. In these experiments, we set $S = 0.02$, $C_M = B$ and $W = 5$. Plots are reported for three different values of N_t. Both figures show that the processing time decreases as the number of miners increases, which is a sign of the good scalability of the architecture. Scalable behavior is ensured by two main factors: the linearity property of the sketch algorithm, and the placement of Data Cachers close to the miners. The system is stable when the execution time is lower than the time period P (15 s): in such a case, the system is able to keep the pace with the generation of stream data. This condition is always verified when the system is only asked to compute frequent items, as is clear from Fig. 4a. On the other hand, the computation of frequent itemsets is much more time consuming. The dashed line depicted in Fig. 4b corresponds to the time period P, and it is shown to easily check in which cases the system is stable.

Table 2 Some frequent itemsets discovered by analyzing the AltaIrpinia dataset, with the corresponding support value

Frequent 4-itemsets
{LeafWetness = 0, SoilMoisture = 10, AtmTemp = 11, SoilTemp = 5} : 12.2 %
{LeafWetness = 0, SoilMoisture = 10, AtmTemp = 11, SoilTemp = 6} : 9.3 %
{LeafWetness = 0, SoilMoisture = 10, AtmTemp = 12, SoilTemp = 5} : 6.4 %
{LeafWetness = 0, SoilMoisture = 10, AtmTemp = 12, SoilTemp = 6} : 11.0 %
{LeafWetness = 0, SoilMoisture = 10, AtmTemp = 12, SoilTemp = 7} : 10.8 %
...
Frequent 3-itemsets
{LeafWetness = 0, SoilMoisture = 10, AtmTemp = 10} : 11.3 %
{LeafWetness = 0, SoilMoisture = 10, AtmTemp = 11} : 32.5 %
{LeafWetness = 0, SoilMoisture = 10, AtmTemp = 12} : 35.1 %
{LeafWetness = 0, SoilMoisture = 10, AtmTemp = 13} : 14.5 %
{LeafWetness = 0, SoilMoisture = 10, SoilTemp = 5} : 25.5 %
{LeafWetness = 0, SoilMoisture = 10, SoilTemp = 6} : 27.9 %
{LeafWetness = 0, SoilMoisture = 10, SoilTemp = 7} : 24.0 %
...
Frequent 2-itemsets
{LeafWetness = 0, SoilMoisture = 1} : 8.0 %
{LeafWetness = 0, SoilMoisture = 10} : 68.5 %
{LeafWetness = 0, SoilMoisture = 2} : 8.4 %
{LeafWetness = 0, SoilMoisture = 100}: 7.5 %
{LeafWetness = 0, SoilMoisture = 3} : 16.4 %
{LeafWetness = 0, SoilMoisture = 9} : 29.1 %
{LeafWetness = 0, AtmTemp = 10} : 16.3 %
{LeafWetness = 0, AtmTemp = 11} : 35.5 %
...

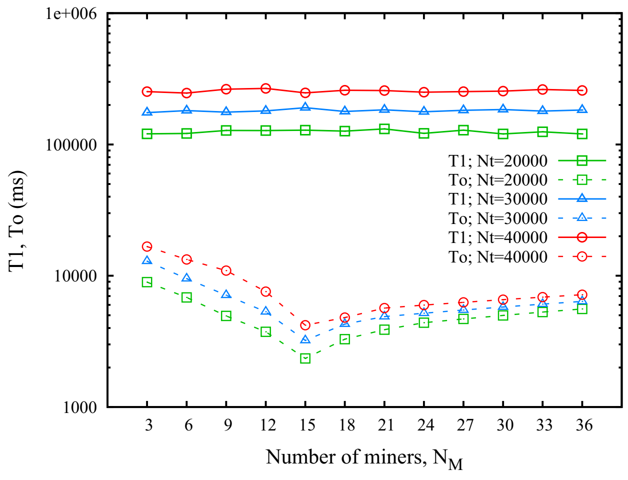


(a) Computation of frequent items (I).

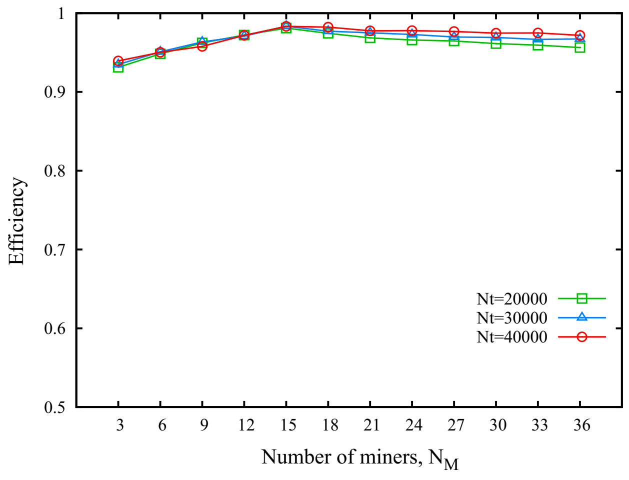
(b) Computation of frequent items (I) and itemsets (IS).

Fig. 4 Execution time for the computation of frequent items (I) and itemsets (IS), vs. the number of miners, for different values of the number of transactions per block, N_t

An efficiency analysis was performed in accordance with the study of parallel architectures presented in Grama et al. (1993). Specifically, we extracted the overall computation time T_C, i.e., the sum of the computation times measured on the different miners, and the overall overhead time T_O, defined as the sum of all the times spent



(a) T_C and T_O



(b) efficiency

Fig. 5 Computing time (T_C) and overhead time (T_O), efficiency vs. the number of miners, for different values of the number of transactions per block, N_t

in other activities, which practically coincide with the transfer times. These two indexes are shown in Fig. 5a using a logarithmic scale. The overall computation time T_C, on the other hand, has a nearly constant trend. The efficiency of the computation can be defined as the fraction of time that the miners actually devote to computation with respect to the sum of computation and overhead time: $E = \frac{T_C}{T_C+T_O}$. Figure 5b reports efficiency values, always higher than 0.9. It is appreciated that efficiency increases as the number of miners per domain increases up to 5, i.e., the window size B. This effect is induced by the use of caching, as explained in Grama et al. (1993). Specifically, when N_{MD} increases up to the window size, each miner needs to request less data from the Data Cacher, because more data can be stored in the local cache: this leads to a higher efficiency. For larger values of N_{MD}, this effect does not hold anymore and the efficiency slightly decreases, being still very high. Thus, we can observe that the architecture maximizes its efficiency for $N_{MD} = B$, because data can be retrieved from the local cache, by minimizing the communication with the Data Cacher. Moreover, it is noticed that the efficiency increases with the rate of data streams. This means that the distributed architecture is increasingly convenient when the problem size increases, which is a another sign of good scalability properties.

6 Conclusion

The distributed stream mining system presented in this paper is aimed at solving the problem of computing frequent items and frequent itemsets from distributed data streams. The experimental results confirm that the approach is scalable and can manage large data production by using an appropriate number of miners in the distributed architecture. As future work, some optimizations in the communication protocol could be studied, to reduce the communication overhead as well as to save energy of the sensor batteries. In addition, the effectiveness of the architecture could be tested on a larger real-world test case.

Acknowledgments This research work has been funded by the project INSYEME.

References

Cesario E, De Caria N, Mastroianni C, Talia D (2009) Distributed data mining using a public resource computing framework. In: Proceedings of the CoreGRID ERCIM Working Group Workshop on Grids, P2P and Service Computing, pp 33–44

Cesario E, Mastroianni C, Talia D (2014) A multi-domain architecture for mining frequent items anditemsets from distributed data streams. J Grid Comput 12(1):153–168

Charikar M, Chen K, Farach-Colton M (2002) Finding frequent items in data streams. In: Proceedings of the 29thInternational Colloquium on Automata, Languages and Programming (ICALP), pp 693–703

Cormode G, Muthukrishnan S (2005) An improved data stream summary: the count-min sketch and its applications. J Algorithms 55(1):58–75

Cormode G, Garofalakis M (2008) Approximate continuous querying over distributed streams. ACM Trans Database Syst 33(2):1–39

Charikar M, Chen K, Farach-Colton M (2002) Finding frequent items in data streams. In: Proceedings of the 29thInternational Colloquium on Automata, Languages and Programming (ICALP), pp 693–703

Cormode G, Hadjieleftheriou M (2009) Finding the frequent items in streams of data. Commun ACM 52(10):97–105

Fischer M, Salzburg S (1982) Finding a majority among n votes: solution to problem. J Algorithms 3(4):376–379

Gaber M, Zaslavsky A, Krishnaswamy S (2005) Mining data streams: a review. ACM SIGMOD Rec 34(1):18–26

Grama AY, Gupta A, Kumar V (1993) Isoefficiency: measuring the scalability of parallel algorithms and architectures. IEEE Parallel Distrib Technol 1(3):12–21

Greenwald M, Khanna S (2001) Space-efficient online computation of quantile summaries. In: Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data (SIGMOD), pp 58–66

Jin R, Agrawal G (2005) An algorithm forin-core frequent itemset mining on streaming data. In: Proceedings of the 5th IEEE International Conference on Data Mining (ICDM), pp 210–217

Jin R, Agrawal G (2005) An algorithm forin-core frequent itemset mining on streaming data. In: Proceedings of the 5th IEEE International Conference on Data Mining (ICDM), pp 210–217

Parthasarathy S, Ghoting A, Otey ME (2007) A survey of distributed mining of data streams. In: Aggarwal C (ed) Data Streams: Models and Algorithms. Springer, pp 289–307

Shrivastava N, Buragohain C, Agrawal D, Suri S (2004) Medians and beyond: New aggregation techniques. In: Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems (SenSys), pp 239–249

Wright A (2010) Data streaming 2.0. Commun ACM 53(4):13–14